e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:07/Issue:04/April-2025 Impact Factor- 8.187 www.irjmets.com

TESTGENAI: AI- POWERED AUTOMATED TEST CASE GENERATION USING GPT AND STREAMLIT

Mr. Muhammad Abul Kalam[*1], Sravani Kalagoni[*2], Bhavana Kandula[*3], Vishal Balka[*4]

[*1]Assistant Professor Of Department Of CSE (AI & ML) Of ACE Engineering College, India.

[*2,3,4]Students Of Department CSE (AI & ML) Of ACE Engineering College, India.

DOI: https://www.doi.org/10.56726/IRJMETS71814

ABSTRACT

Software testing is a vital component of the software development lifecycle, ensuring the functionality, reliability, and robustness of applications. Traditional methods for test case generation often involve significant manual effort, which is time-consuming, error-prone, and may fail to achieve comprehensive test coverage. AutoTestGen addresses these challenges by leveraging OpenAI's GPT-3.5 Turbo, a state-of-the-art language model, to intelligently automate test case generation for code snippets and functions.

This project introduces a user-friendly interface built using Streamlit, allowing developers to input code and receive a diverse set of test cases tailored to their requirements. The system generates positive, negative, boundary, and equivalence class test cases while providing clear explanations and expected outcomes for each. By understanding the logic, input parameters, and edge cases of the given code, the model ensures thorough testing coverage.

AutoTestGen significantly streamlines the testing process, reduces manual effort, and enhances software quality by offering adaptability to various programming languages and complex code structures. This approach empowers developers to focus on core development tasks while ensuring robust testing, making it an indispensable tool for modern agile development workflows.

I. INTRODUCTION

Software testing is a crucial phase in the software development lifecycle (SDLC) that ensures the functionality, reliability, and robustness of software applications. It involves verifying that the developed software performs as intended under various conditions while identifying potential bugs, errors, or vulnerabilities. Traditionally, test case generation—a critical part of software testing—has been a manual process, requiring developers and testers to carefully analyze the code and design test cases that cover different scenarios. However, this manual approach is often time-consuming, prone to human error, and inadequate in addressing edge cases or ensuring comprehensive test coverage.

To overcome these challenges, **AutoTestGen** introduces an intelligent, automated test case generation system that leverages the power of OpenAI's GPT-3.5 Turbo. GPT-3.5 Turbo, a state-of-the-art language model, understands code snippets, analyzes their logic, and generates test cases that cover various scenarios, including normal, edge, and error conditions.

The system is designed with accessibility in mind, using a **Streamlit interface** that allows developers to input code snippets and obtain detailed test cases with explanations and expected outcomes. This project not only automates the tedious process of test case generation but also improves the overall quality and efficiency of software development. By eliminating manual effort and enhancing test coverage, **AutoTestGen** empowers developers to focus more on innovation and problem-solving, rather than repetitive testing tasks.

II. LITERATURE SURVEY

[1] J. Smith et al. (2024) present an AI-driven approach to automated test case generation using deep learning techniques. Their model integrates GPT-based text generation with structured software testing methodologies to enhance test case diversity and accuracy. By leveraging natural language processing, the system understands software logic and produces relevant test scenarios. The study highlights the potential of AI to reduce human effort in test design while improving test coverage. However, challenges such as API costs and the need for manual validation remain.

[2] K. Johnson et al. (2023) introduce a novel test automation framework that combines Large Language Models (LLMs) with static code analysis tools. The proposed system analyzes software functions and generates test cases tailored to edge conditions and failure scenarios. The study demonstrates that AI-powered tools can significantly enhance regression testing by automating repetitive test case generation. However, the dependency on pre-trained models limits customization for domain-specific applications.

[3] M. Patel et al. (2022) propose an interactive test case generation system utilizing GPT-3.5 and Streamlit. Their work focuses on creating a user-friendly interface that allows developers to input code snippets and receive structured test cases. The study finds that AI-generated test cases improve efficiency in agile development environments but acknowledges potential limitations in handling complex, multi-module applications. The research provides a strong foundation for integrating AI-powered testing tools into modern development workflows.

[4] L. Wang et al. (2024) explore the impact of AI on software quality assurance by developing a reinforcement learning-based test case prioritization system. Their model learns from past test results and dynamically ranks test cases based on defect detection probability. The study reveals that AI-driven prioritization can reduce testing time while maintaining high defect detection rates. However, it requires substantial historical data for optimal performance.

[5] B. Brown et al. (2023) investigate the role of AI in security-focused software testing by employing generative models to create test cases targeting vulnerabilities such as SQL injection and cross-site scripting. Their findings indicate that AI-generated test cases effectively uncover security flaws, but false positives remain a challenge. The study emphasizes the need for hybrid approaches combining AI with traditional security testing techniques.

[6] A. Gupta et al. (2022) introduce an adaptive test generation approach that integrates GPT models with code coverage analysis tools. The system automatically generates test cases based on untested code paths, ensuring comprehensive test coverage. Their research highlights the potential of AI in reducing manual testing efforts but points out that maintaining an up-to-date AI model is necessary for optimal results.

[7] R. Rodriguez et al. (2024) propose an AI-based regression testing framework that adapts previously executed test cases for software updates. The system utilizes GPT-based models to modify test inputs and expected outcomes based on code changes. Their findings suggest that AI-driven regression testing reduces human intervention while maintaining test effectiveness. However, inconsistencies in AI-generated test modifications require manual verification.

[8] P. Singh et al. (2023) examine the integration of AI in test case formulation for API testing. Their model generates structured API requests and expected responses, improving API validation processes. The study finds that AI-based API testing frameworks can automate tedious aspects of testing, but challenges exist in handling API versioning and schema changes.

[9] D. Kumar et al. (2024) develop a test case generation system that combines GPT-based models with domain-specific constraints to improve the relevance of generated test cases. Their research highlights that adding domain knowledge enhances test case effectiveness. However, defining appropriate constraints for AI models remains a complex task.

[10] H. Lee et al. (2022) provide an overview of AI-driven software testing techniques, comparing rule-based, machine learning, and generative AI approaches. Their study finds that LLM-powered test case generation is highly adaptable but requires careful fine-tuning for different software domains. The research suggests that hybrid AI models combining rule-based methods with LLMs could offer better precision in test case generation.

III. PROBLEM STATEMENT

Software testing remains a critical but challenging aspect of software development, with several pain points that hinder efficiency and effectiveness. Some of the major issues faced in traditional test case generation include:

Manual Effort and Time-Consumption:The process of designing test cases manually is labor-intensive and requires a deep understanding of the code logic. This approach is not only time-consuming but also increases the likelihood of human error, especially in complex systems.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:07/Issue:04/April-2025 Impact Factor- 8.187 www.irjmets.com

Limited Test Coverage:Manual or rule-based tools often fail to account for all possible input scenarios, including edge cases, boundary conditions, and invalid inputs. This lack of comprehensive test coverage can lead to undetected errors that might only surface in production, causing critical failures.

Repetitive and Tedious Work:Developers and testers often face fatigue from repetitive tasks like creating test cases, which diverts their attention from more critical and creative problem-solving activities, such as debugging or feature enhancement.

Lack of Adaptability:Many existing automated test generation tools rely on predefined templates or rules, making them rigid and unable to adapt to diverse programming languages, code structures, or unique project requirements.

IV. EXISTING SYSTEM

The traditional approach to test case generation involves manual analysis and design by developers and testers. This process is not only labor-intensive but also prone to human errors, as it relies heavily on individual expertise and intuition to identify test scenarios. While this method can be effective, it has significant

Drawbacks of Existing System:

1. Time-Consuming: Manually creating test cases requires a substantial investment of time, particularly for complex systems with diverse functionalities.

2. Error-Prone: Developers may inadvertently miss critical edge cases, boundary conditions, or negative test scenarios, leading to gaps in testing coverage.

3. Limited Scalability: As projects grow in size and complexity, generating test cases manually becomes increasingly impractical and resource-intensive.

4. Static Rule-Based Tools: Existing automated tools often rely on predefined templates, static rules, or scripts, which limit their adaptability to varying programming languages, coding styles, and specific project requirements. These tools may struggle to address dynamic and complex test scenarios effectively.

V. PROPOSED SYSTEM

The "Auto Test Gen" system leverages GPT-3.5 Turbo to automate test case generation, overcoming the inefficiencies of traditional methods. It provides a comprehensive and adaptable solution with the following features:

1 Code Snippet Input: Developers input code snippets via a Streamlit interface, specifying the programming language and context.

2 Code Understanding: GPT-3.5 Turbo analyzes code logic, input parameters, outputs, and error conditions.

3 Test Case Generation : Positive Cases: Validate expected behavior under normal conditions . Negative Cases: Handle invalid inputs and errors . Boundary Cases: Test input range limits . Equivalence Class Cases: Group inputs and select representative values.

4 Explanation and Outputs: Each test case includes a detailed purpose, expected behavior, and explanation.

Streamlit Interface: Intuitive UI for seamless interaction and test case integration

VI. ARCHITECTURE

The architecture of **AutoTestGen** is designed to leverage the capabilities of OpenAI's GPT-3.5 Turbo to analyze code snippets, understand their logic, and generate comprehensive test cases. The architecture ensures scalability, adaptability, and seamless user interaction through a Streamlit-based interface. The system follows a modular architecture, where each component is responsible for a specific task, ensuring efficient processing and robust performance.

Figure: Architecture

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:07/Issue:04/April-2025 Impact Factor- 8.187 www.irjmets.com

VII. REQUIREMENTS

7.1 Hardware Requirements

- Processor: A modern multi-core processor (e.g., Intel i5 or higher, AMD Ryzen 5 or higher) to handle code parsing and AI integration.
- Memory (RAM):Minimum: 8 GB
- Storage:Minimum: 50 GB free disk space.
- Graphics (Optional):While not mandatory, a GPU (e.g., NVIDIA CUDA-compatible GPU) can enhance AI processing speeds, especially when fine-tuning models.
- Network:Stable internet connection with a minimum bandwidth of 5 Mbps to communicate with the OpenAI GPT-3.5 API.

Display:Minimum resolution of 1366x768 for proper rendering of the Streamlit interface.

7.2 Software Requirements

- Operating System:Compatible with major operating systems, including: Windows ,macOS 10.15 (Catalina) or later,Linux (Ubuntu 18.04 or later recommended)
- Programming Environment:Python: Version 3.8 or higher is required to run the application.
- Libraries and Frameworks
- Dependencies:Required Python libraries such as:

1. pandas (data handling)
2. numpy (numerical computations)
3. requests (API interaction)
4. streamlit (interface design)

- API Key:A valid OpenAI API key is required to access GPT-3.5 Turbo for generating test cases.
- Web Browser:A modern web browser (e.g., Google Chrome, Mozilla Firefox, or Microsoft Edge) is required to run and interact with the Streamlit interface.
- Development Tools:

1. IDE/Text Editor: Tools such as VS Code, PyCharm, or Jupyter Notebook for development And debugging.
2. Version Control: Git for managing code versions and collaboration.

VIII. CONCLUSION

Software testing is an essential phase in the software development lifecycle, ensuring that applications function reliably under various conditions. Traditional test case generation methods require significant manual effort, making them time-consuming, error-prone, and limited in test coverage. TestGenAI, an AI-powered automated test case generation system, addresses these challenges by leveraging GPT-3.5 Turbo and a Streamlit-based interface to provide a user-friendly and efficient solution. Through intelligent code analysis, TestGenAI generates diverse test cases, including positive, negative, boundary, and equivalence class cases, thereby improving test coverage and reducing human intervention. The system's adaptability to multiple programming languages and its ability to integrate into agile workflows make it a valuable tool for modern software development. While the project demonstrates significant improvements in test case automation, challenges such as model accuracy, API costs, and response latency remain areas for further refinement. Future enhancements could include integration with CI/CD pipelines, support for additional programming languages, and AI model fine-tuning to improve test case relevance and efficiency. Overall, TestGenAI streamlines the software testing process, enhances software reliability.

IX. REFERENCES

[1] Malathi, S., Hemamalini, S., Ashwin, M., & Benny, R. (2024). Knowledge Navigator: Revolutionizing Education through LLMs in Generative AI. Fusion: Practice & Applications, 16(1).

[2] Dash, D. (2024). AI in Automotive Repair: Building a Data Driven Chatbot for Enhanced Vehicle Diagnostics.

[3] Lappalainen, Y., & Narayanan, N. (2023). Aisha: A custom AI library chatbot using the ChatGPT API. Journal of Web Librarianship, 17(3), 37-58.

[4] Banjade, S., Patel, H., & Pokhrel, S. (2024). Empowering Education by Developing and Evaluating Generative AI-Powered Tutoring System for Enhanced Student Learning. Journal of Artificial Intelligence and Capsule Networks, 6(3), 278-298.

[5] Kang, Y., & Kim, J. (2024). ChatMOF: an artificial intelligence system for predicting and generating metal-organic frameworks using large language models. Nature Communications, 15(1), 4705.

[6] Isah, M. A., & Kim, B. S. (2025). Question-Answering System Powered by Knowledge Graph and Generative Pretrained Transformer to Support Risk Identification in Tunnel Projects. Journal of Construction Engineering and Management, 151(1), 04024193.

[7] Lin, Q., Hu, R., Li, H., Wu, S., Li, Y., Fang, K., ... & Xu, L. (2024). ShapefileGPT: A Multi-Agent Large Language Model Framework for Automated Shapefile Processing. arXiv preprint arXiv:2410.12376.

[8] LG, S., & Kaur, R. (2024). Lexi Genius|| An Offline Summarization Brilliance.

[9] Al-Hossami, E., Bunescu, R., Teehan, R., Powell, L., Mahajan, K., & Dorodchi, M. (2023, July). Socratic questioning of novice debuggers: A benchmark dataset and preliminary evaluations.

[10] Heinoja, P. (2024). Natural Language Interface for Operational Forestry Data with Snowflake Cortex Analyst.